ABC-Boost

Adaptive Base Class Boost for Multi-class Classification

Ping Li

Department of Statistical Science

Faculty of Computing and Information Science

Cornell University

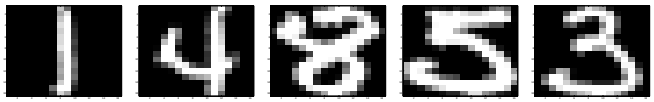
Ithaca, NY 14853

June 17, 2009

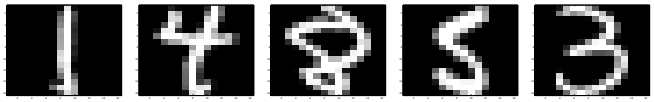
What is (Multi-class) Classification?

An Example: USPS Handwritten Zipcode Recognition

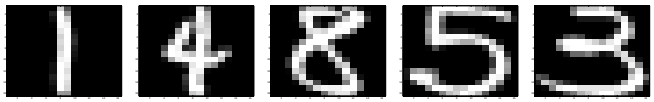
Person 1:



Person 2:



Person 3:



10-class classification problem

Multi-Class Classification

Given a training data set

$$\{y_i,\ X_i\}_{i=1}^{N},\qquad X_i \in \mathbb{R}^p,\qquad y_i \in \{0, 1, 2, ..., K-1\}$$

the task is to learn a function which predicts the class label y_i from X_i.

- $K = 2$: binary classification
- $K > 2$: multi-class classification (e.g., $K = 5$ in search)

One Strategy for Multi-class Classification

Learn the class probabilities

$$\hat{p}_k = \mathbf{Pr}\left\{y = k|X\right\}, \qquad k = 0, 1, ..., K-1,$$

$$\sum_{k=0}^{K-1} \hat{p}_k = 1, \qquad \left(\text{only } K-1 \text{ degrees of freedom}\right).$$

Assign the class label according to

$$\hat{y}|X = \underset{k}{\text{argmax}}\ \hat{p}_k$$

The Multinomial Logit Probability Model

$$p_k = \frac{e^{F_k}}{\sum_{s=0}^{K-1} e^{F_s}}$$

where $F_k = F_k(X)$. Logistic regression: $F_k(X) = \beta^{\mathsf{T}} X$.

The sum-to-zero constraint

$$\sum_{k=0}^{K-1} F_k(X) = 0$$

is commonly used to obtain a unique solution (only $K - 1$ degrees of freedom).

Why the sum-to-zero constraint?

$$\frac{e^{F_{i,k}+C}}{\sum_{s=0}^{K-1} e^{F_{i,s}+C}} = \frac{e^C e^{F_{i,k}}}{e^C \sum_{s=0}^{K-1} e^{F_{i,s}}} = \frac{e^{F_{i,k}}}{\sum_{s=0}^{K-1} e^{F_{i,s}}} = p_{i,k}.$$

For identifiability, one should impose a constraint. One popular (and natural) choice is to assume $\sum_{k=0}^{K-1} F_{i,k} = const$, which is equivalent to

$$\sum_{k=0}^{K-1} F_{i,k} = 0.$$

Learning by multinomial maximum likelihood

Seek $F_{i,k}$ to maximize the mutlinomial likelihood: Suppose $y_i = k$,

$$Lik \propto p_{i,0}^0 \times ... \times {p_{i,k}}^1 \times ... \times p_{i,K-1}^0 = p_{i,k}$$

or equivalently, maximizing the log likelihood:

$$\log Lik \propto \log p_{i,k}$$

Or equivalently, minimizing the negative log likelihood loss

$$L_i = -\log p_{i,k}$$

In logistic regression: $F_k(X) = \beta^{\mathsf{T}} X$, the task is to learn β.

The negative log-likelihood loss

$$L = \sum_{i=1}^{N} L_i = \sum_{i=1}^{N} \left\{ - \sum_{k=0}^{K-1} r_{i,k} \log p_{i,k} \right\}$$

$$r_{i,k} = \begin{cases} 1 & \text{if } y_i = k \\ 0 & \text{otherwise} \end{cases}$$

Data set $\{y_i,\ X_i\}_{i=1}^{N}$, $\quad X_i \in \mathbb{R}^p$, $\quad y_i \in \{0, 1, 2, ..., K-1\}$

Singularity of Hessian without Sum-to-zero Constraint

Without sum-to-zero constraint $\sum_{k=0}^{K=1} F_{i,k} = 0$:

$$\frac{\partial L_i}{\partial F_{i,k}} = -\left(r_{i,k} - p_{i,k}\right), \qquad \frac{\partial^2 L_i}{\partial F_{i,k}^2} = p_{i,k}\left(1 - p_{i,k}\right).$$

For example, when $K = 3$.

$$\left|\begin{array}{ccc} \frac{\partial^2 L_i}{\partial p_0^2} & \frac{\partial^2 L_i}{\partial p_0 p_1} & \frac{\partial^2 L_i}{\partial p_0 p_2} \\ \frac{\partial^2 L_i}{\partial p_1 p_0} & \frac{\partial^2 L_i}{\partial p_1^2} & \frac{\partial^2 L_i}{\partial p_1 p_2} \\ \frac{\partial^2 L_i}{\partial p_2 p_0} & \frac{\partial^2 L_i}{\partial p_2 p_1} & \frac{\partial^2 L_i}{\partial p_2^2} \end{array}\right| = \left|\begin{array}{ccc} p_0(1-p_0) & -p_0 p_1 & -p_0 p_2 \\ -p_1 p_0 & p_1(1-p_1) & -p_1 p_2 \\ -p_2 p_0 & -p_2 p_1 & p_2(1-p_2) \end{array}\right| = 0$$

Diagonal Approximation

(Friedman et. al, 2000, Friedman 2001) used diagonal approximation of Hessian:

$$\frac{K-1}{K}\begin{bmatrix} \frac{\partial^2 L_i}{\partial p_0^2} & & \\ & \frac{\partial^2 L_i}{\partial p_1^2} & \\ & & \frac{\partial^2 L_i}{\partial p_2^2} \end{bmatrix}$$

$$=\frac{K-1}{K}\begin{bmatrix} p_0(1-p_0) & & \\ & p_1(1-p_1) & \\ & & p_2(1-p_2) \end{bmatrix}$$

Function Gradient Boosting

Much more flexible and accurate than logistic regression.

$\{y_i, \mathbf{x}_i\}_{i=1}^N$: training data set; L: a differentiable loss function.

A "greedy stagewise" approach builds an additive function $F^{(M)}$

$$F^{(M)}(\mathbf{x}) = \sum_{m=1}^{M} \rho_m h(\mathbf{x}; \mathbf{a}_m),$$

such that, at each stage m, $m = 1$ to M,

$$\{\rho_m, \mathbf{a}_m\} = \underset{\rho, \mathbf{a}}{\text{argmin}} \sum_{i=1}^{N} L\left(y_i, F^{(m-1)}(\mathbf{x}_i; \mathbf{a}, \rho)\right).$$

$h(\mathbf{x}; \mathbf{a})$ is the weak learner (e.g., regression trees).

(Friedman 2001) approximately conducted steepest descent in the function space, by solving a least square problem

$$\mathbf{a}_m = \underset{\mathbf{a},\rho}{\text{argmin}} \sum_{i=1}^{N} \left[-g_m(\mathbf{x}_i) - \rho h(\mathbf{x}_i; \mathbf{a})\right]^2,$$

where

$$-g_m(\mathbf{x}_i) = -\left[\frac{\partial L\left(y_i, F(\mathbf{x}_i)\right)}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F^{(m-1)}(\mathbf{x})}$$

is the steepest descent direction in the N-dimensional data space.

For ρ_m, a line search is performed:

$$\rho_m = \underset{\rho}{\text{argmin}} \sum_{i=1}^{N} L\left(y_i, F^{(m-1)}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m)\right).$$

Alg. 1: the Generic Gradient Boosting Algorithm

1: $F_{\mathbf{x}} = \text{argmin}_{\rho} \sum_{i=1}^{N} L(y_i, \rho)$

2: For $m = 1$ to M Do

3: $\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F^{(m-1)}(\mathbf{x})}, i = 1 \text{ to } N.$

4: $\mathbf{a}_m = \text{argmin}_{\mathbf{a},\rho} \sum_{i=1}^{N} \left[\tilde{y}_i - \rho h(\mathbf{x}_i; \mathbf{a})\right]^2$

5: $\rho_m = \text{argmin}_{\rho} \sum_{i=1}^{N} L\left(y_i, F^{(m-1)}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m)\right)$

6: $F_{\mathbf{x}} = F_{\mathbf{x}} + \rho_m h(\mathbf{x}; \mathbf{a}_m)$

7: End

MART = Gradient Boosting + Regression Trees.

MART (Multiple Additive Regression Trees)

MART = Gradient Boosting + regression trees

1: $F_{i,k} = 0$, $k = 0$ to $K-1$, $i = 1$ to N

2: For $m = 1$ to M Do

3: For $k = 0$ to $K-1$ Do

4: $p_{i,k} = \exp(F_{i,k}) / \sum_{s=0}^{K-1} \exp(F_{i,s})$

5: $\{R_{j,k,m}\}_{j=1}^{J} = J$-terminal node regression tree from $\{r_{i,k} - p_{i,k},\ \mathbf{x}_i\}_{i=1}^{N}$

6: $\beta_{j,k,m} = \frac{K-1}{K} \frac{\sum_{\mathbf{x}_i \in R_{j,k,m}} r_{i,k} - p_{i,k}}{\sum_{\mathbf{x}_i \in R_{j,k,m}} (1-p_{i,k}) p_{i,k}}$

7: $F_{i,k} = F_{i,k} + \nu \sum_{j=1}^{J} \beta_{j,k,m} 1_{\mathbf{x}_i \in R_{j,k,m}}$

8: End

9: End

Key components of MART

- Loss function: $L_i = -\sum_{k=0}^{K-1} r_{i,k} \log p_{i,k}$.
- The first derivative $-\frac{\partial L_i}{\partial F_k} = r_{i,k} - p_{i,k}$ is used to learn the structure (split location/values etc) of the trees.
- The second derivative $\frac{\partial^2 L_i}{\partial F_k^2} = p_{i,k}\left(1 - p_{i,k}\right)$ is used for determining the values of the terminal nodes.
- A factor $\frac{K-1}{K}$ for considering only $K-1$ degrees of freedom.
- A shrinkage factor ν (eg ≤ 0.1) to avoid overfitting.

Derivatives Under Sum-to-zero Constraint

The loss function:

$$L_i = -\sum_{k=0}^{K-1} r_{i,k} \log p_{i,k}$$

The probability model and sum-to-zero constraint:

$$p_{i,k} = \frac{e^{F_{i,k}}}{\sum_{s=0}^{K-1} e^{F_{i,s}}}, \qquad \sum_{k=0}^{K-1} F_{i,k} = 0$$

Without loss of generality, we assume $k = 0$ is the base class

$$F_{i,0} = -\sum_{i=1}^{K-1} F_{i,k}$$

$$\frac{\partial L_i}{\partial F_{i,k}} = (r_{i,0} - p_{i,0}) - (r_{i,k} - p_{i,k}),$$

$$\frac{\partial^2 L_i}{\partial F_{i,k}^2} = p_{i,0}(1 - p_{i,0}) + p_{i,k}(1 - p_{i,k}) + 2p_{i,0}p_{i,k}.$$

Adaptive Base Class Boost (ABC-Boost)

Two Key Ideas of ABC-Boost:

1. Formulate the multi-class boosting algorithm by considering a base class.

   Do not have to train for the base class, which is inferred from the sum-zero-constraint $\sum_{k=0}^{K-1} F_{i,k} = 0$.

2. At each boosting step, adaptively choose the base class.

How to choose the base class?

- We should choose the base class according to performance (training loss).
- (One of many ideas) Exhaustively search for all K base classes and choose the base class that leads to the best performance (smallest training loss).
  - Computationally expensive (but not too bad, unless K is really large)
  - Good performance can be achieved.
- Many other ideas.

ABC-MART = ABC-Boost + MART.

Alg. 3: ABC-MART, simple modification of MART

1: $F_{i,k} = 0$, $k = 0$ to $K-1$, $i = 1$ to N

2: For $m = 1$ to M Do

: For $b = 0$ to $K-1$ DO

3: For $k = 0$ to $K-1$ (and $k \neq b$) Do

4: $p_{i,k} = \exp(F_{i,k}) / \sum_{s=0}^{K-1} \exp(F_{i,s})$

5: $\{R_{j,k,m}\}_{j=1}^{J} = J$-terminal node tree from $\{r_{i,k} - p_{i,k} - (r_{i,b} - p_{i,b}),\ \mathbf{x}_i\}_{i=1}^{N}$

6: $$\beta_{j,k,m} = \frac{\sum_{\mathbf{x}_i \in R_{j,k,m}} (r_{i,k} - p_{i,k}) - (r_{i,b} - p_{i,b})}{\sum_{\mathbf{x}_i \in R_{j,k,m}} (1-p_{i,k}) p_{i,k} + (1-p_{i,b}) p_{i,b} + 2 p_{i,k} p_{i,b}}$$

7: $F_{i,k} = F_{i,k} + \nu \sum_{j=1}^{J} \beta_{j,k,m} 1_{\mathbf{x}_i \in R_{j,k,m}}$

8: End

9: End

Experiments: Data Sets

Data set	K	# training samples	# test samples	# features
Poker	10	25100	1000000	25
Mnist10k	10	10000	60000	784
Covertype	7	290506	290506	54
Letter	26	16000	4000	16
Letter2k	26	2000	18000	16
Letter4k	26	4000	16000	16
Pendigits	10	7494	3498	16
Zipcode	10	7291	2007	256
Optdigits	10	3823	1797	64
Isolet	26	6218	1559	617

Experiments: Summary of Test Misclassification Errors

Data set	MART	ABC-MART	R_{err} (%)	P-value	# test
Poker	86508	69171	20.4	0	1000000
Mnist10k	11439	10375	9.3	0	60000
Covertype	11350	10402	8.2	0	290506
Letter	129	99	23.3	0.02	4000
Letter2k	2439	2180	10.6	0	18000
Letter4k	1352	1126	16.7	0	16000
Pendigits	124	100	19.4	0.05	3498
Zipcode	111	100	9.9	0.22	2007
Optdigits	55	43	21.8	0.11	1797
Isolet	80	64	20.0	0.09	1559

Relative error improvement

$$R_{err} = \frac{\text{classification errors of MART} - \text{classification errors of ABC-MART}}{\text{classification errors of MART}}.$$

Questions and Short Answers

1. Is the improvement due to the particular choice of parameters, J, ν M?

   We experimented with a series of parameters, $\nu \in \{0.04, 0.06, 0.08, 0.1\}$, $J \in \{4, 6, 8, 10, 12, 14, 16, 18, 20\}$, and $M = 10000$ (at most).

2. Is the improvement due to incompetent implementation of MART?

   We also report results obtained by using Friedman's MART program.

3. Is the computational efficiency an issue?

   Training is slower. But, testing is faster.

Experiments: Letter Data Set

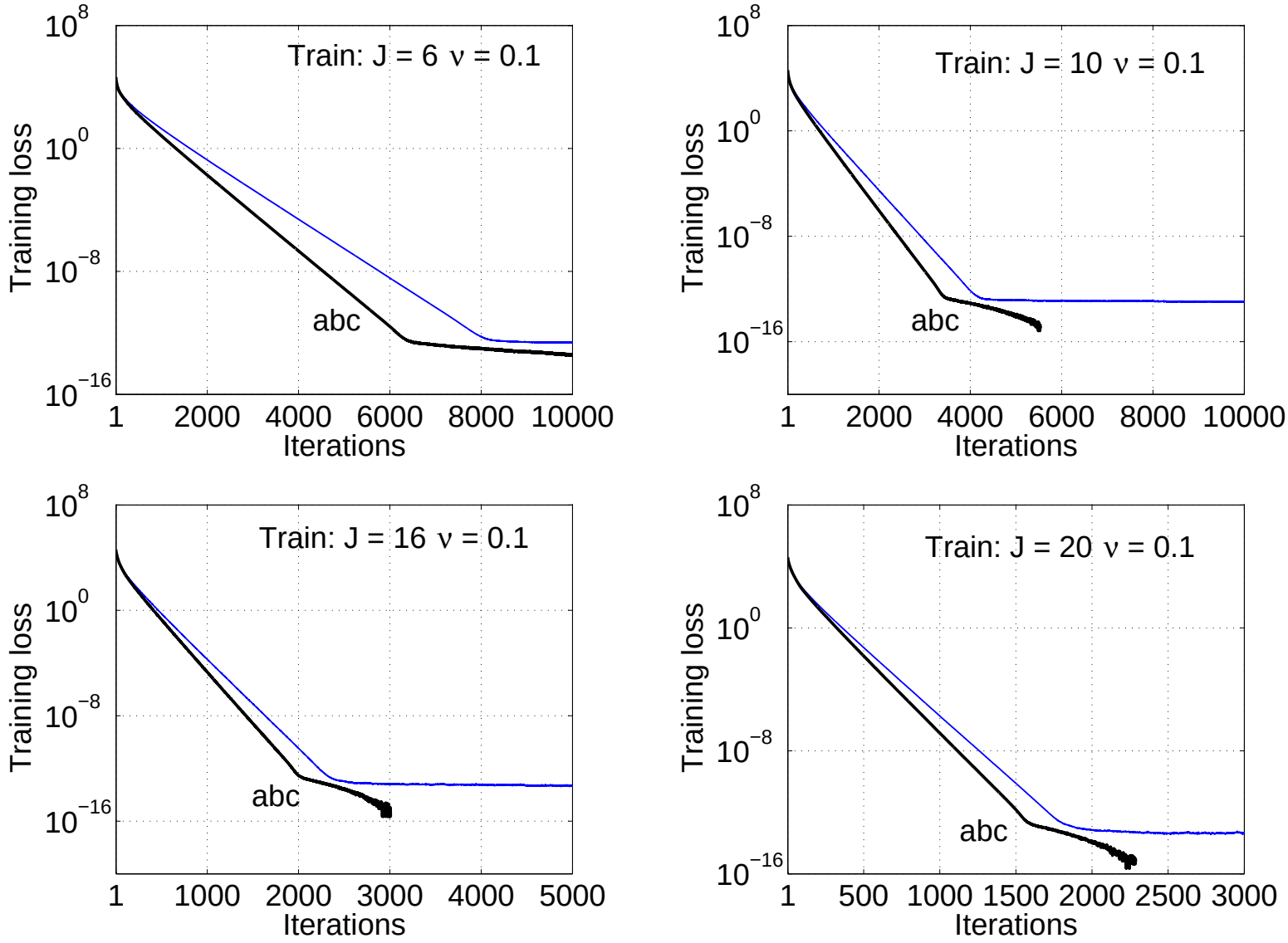
	mart			
	$\nu = 0.04$	$\nu = 0.06$	$\nu = 0.08$	$\nu = 0.1$
$J = 4$	174 [178]	177 [176]	177 [177]	172 [177]
$J = 6$	163 [153]	157 [160]	159 [156]	159 [162]
$J = 8$	155 [151]	148 [152]	155 [148]	144 [151]
$J = 10$	145 [141]	145 [148]	136 [144]	142 [136]
$J = 12$	143 [142]	147 [143]	139 [145]	141 [145]
$J = 14$	141 [151]	144 [150]	145 [144]	152 [142]
$J = 16$	143 [148]	145 [146]	139 [145]	141 [137]
$J = 18$	132 [132]	133 [137]	129 [135]	138 [134]
$J = 20$	129 [143]	140 [135]	134 [139]	136 [143]

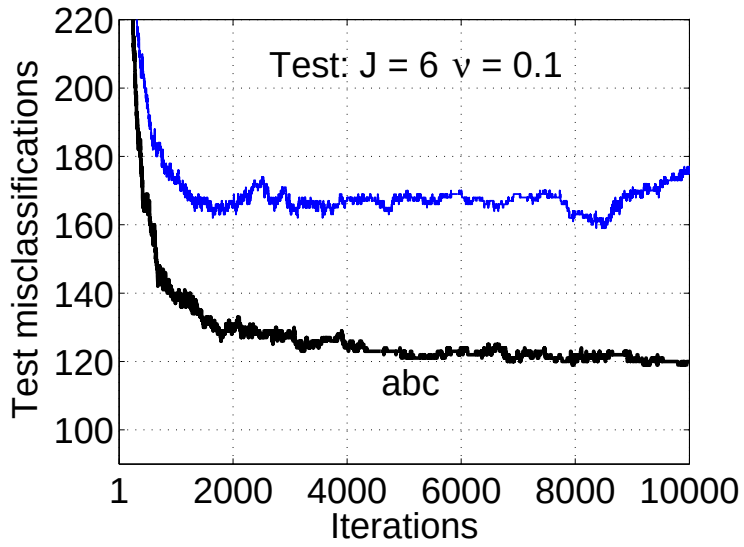
[143] = result from Friedman's MART program.

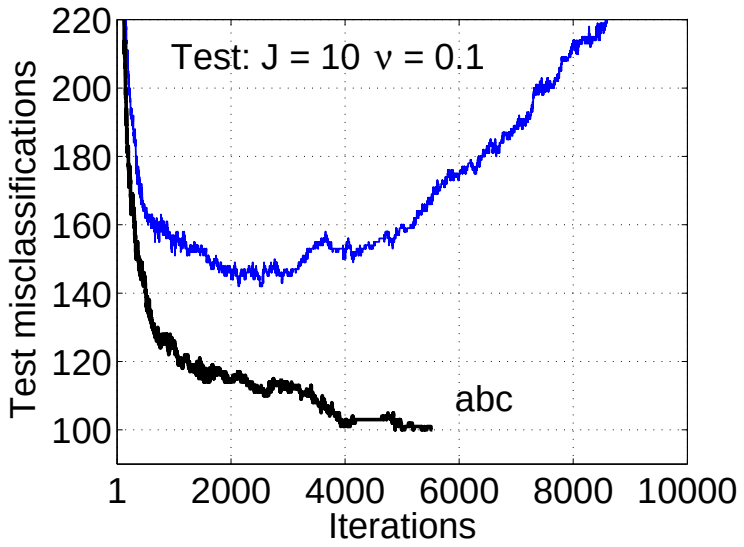
Experiments: Letter Data Set

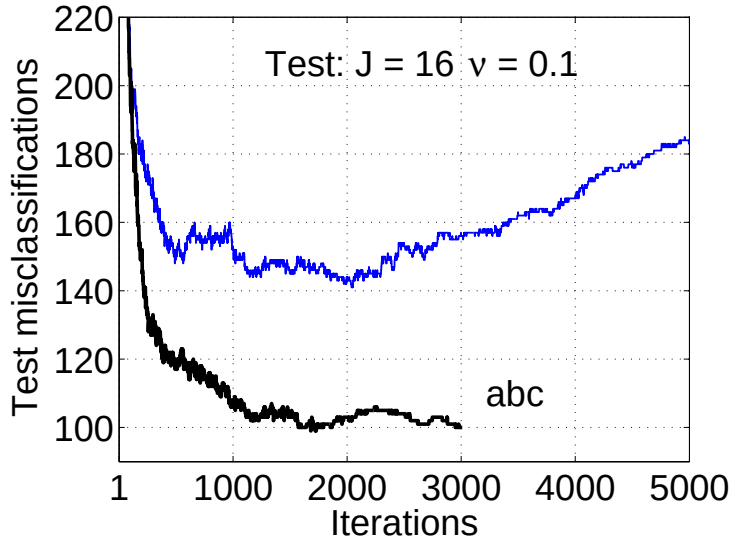
	abc-mart			
	$\nu = 0.04$	$\nu = 0.06$	$\nu = 0.08$	$\nu = 0.1$
$J = 4$	152 (12.6)	147 (16.9)	142 (19.8)	137 (20.3)
$J = 6$	127 (22.1)	126 (19.7)	118 (25.8)	119 (25.2)
$J = 8$	122 (21.3)	112 (24.3)	108 (30.3)	103 (28.5)
$J = 10$	126 (13.1)	115 (20.7)	106 (22.1)	100 (29.6)
$J = 12$	117 (18.2)	114 (22.4)	107 (23.0)	104 (26.2)
$J = 14$	112 (20.6)	113 (21.5)	106 (26.9)	108 (28.9)
$J = 16$	111 (22.4)	112 (22.8)	106 (23.7)	99 (29.8)
$J = 18$	113 (14.4)	110 (17.3)	108 (16.3)	104 (24.6)
$J = 20$	100 (22.5)	104 (25.7)	100 (25.4)	102 (25.0)

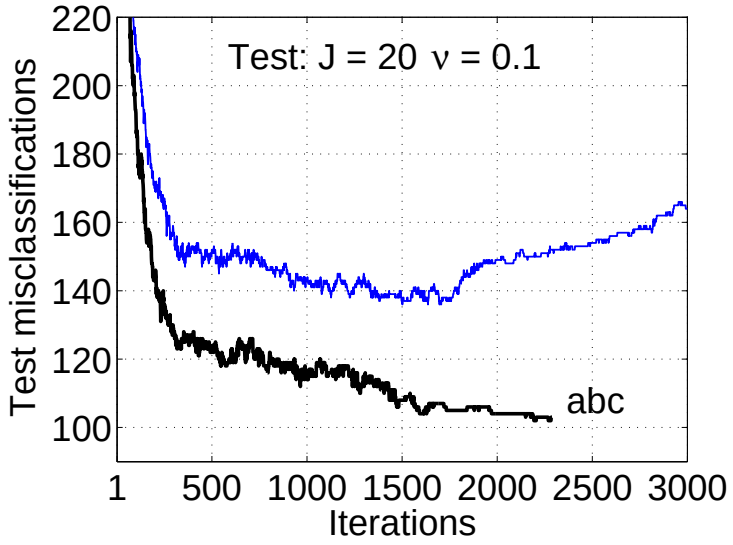
(22.5) = relative improvement R_{err} (%).


Train: J = 6 ν = 0.1
Training loss
Iterations
abc
Train: J = 10 ν = 0.1
Training loss
Iterations
abc
Train: J = 16 ν = 0.1
Training loss
Iterations
abc
Train: J = 20 ν = 0.1
Training loss
Iterations
abc


Test: J = 6 ν = 0.1
abc
Test misclassifications
Iterations
220
200
180
160
140
120
100
1
2000
4000
6000
8000
10000


Test: J = 10 ν = 0.1
abc
Test misclassifications
Iterations
220
200
180
160
140
120
100
1
2000
4000
6000
8000
10000


Test: J = 16 ν = 0.1
abc
Test misclassifications
Iterations
220
200
180
160
140
120
100
1
1000
2000
3000
4000
5000

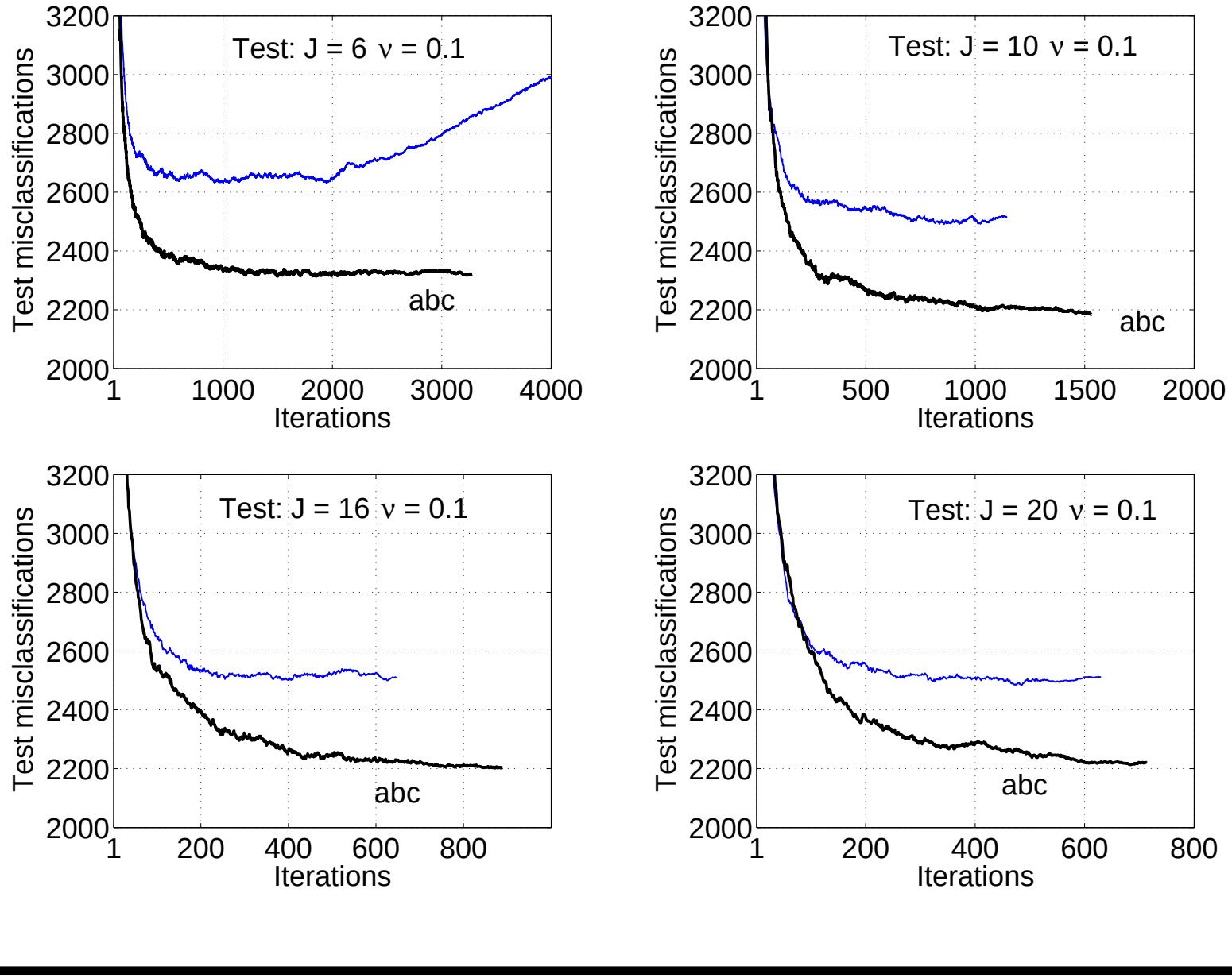

Test: J = 20 ν = 0.1
abc
Test misclassifications
Iterations
220
200
180
160
140
120
100
1
500
1000
1500
2000
2500
3000

Experiments: Letter2k Dataset

	mart			
	$\nu = 0.04$	$\nu = 0.06$	$\nu = 0.08$	$\nu = 0.1$
$J = 4$	2694 [2750]	2698 [2728]	2684 [2706]	2689 [2733]
$J = 6$	2683 [2720]	2664 [2688]	2640 [2716]	2629 [2688]
$J = 8$	2569 [2577]	2603 [2579]	2563 [2603]	2571 [2559]
$J = 10$	2534 [2545]	2516 [2546]	2504 [2539]	2491 [2514]
$J = 12$	2503 [2474]	2516 [2465]	2473 [2492]	2492 [2455]
$J = 14$	2488 [2432]	2467 [2482]	2460 [2451]	2460 [2454]
$J = 16$	2503 [2499]	2501 [2494]	2496 [2437]	2500 [2424]
$J = 18$	2494 [2464]	2497 [2482]	2472 [2489]	2439 [2476]
$J = 20$	2499 [2507]	2512 [2523]	2504 [2460]	2482 [2505]

Experiments: Letter2k Dataset

	abc-mart			
	$\nu = 0.04$	$\nu = 0.06$	$\nu = 0.08$	$\nu = 0.1$
$J = 4$	2476 (8.1)	2458 (8.9)	2406 (10.4)	2407 (10.5)
$J = 6$	2355 (12.2)	2319 (12.9)	2309 (12.5)	2314 (12.0)
$J = 8$	2277 (11.4)	2281 (12.4)	2253 (12.1)	2241 (12.8)
$J = 10$	2236 (11.8)	2204 (12.4)	2190 (12.5)	2184 (12.3)
$J = 12$	2199 (12.1)	2210 (12.2)	2193 (11.3)	2200 (11.7)
$J = 14$	2202 (11.5)	2218 (10.1)	2198 (10.7)	2180 (11.4)
$J = 16$	2215 (11.5)	2216 (11.4)	2228 (10.7)	2202 (11.9)
$J = 18$	2216 (11.1)	2208 (11.6)	2205 (10.8)	2213 (9.3)
$J = 20$	2199 (12.0)	2195 (12.6)	2183 (12.8)	2213 (10.8)

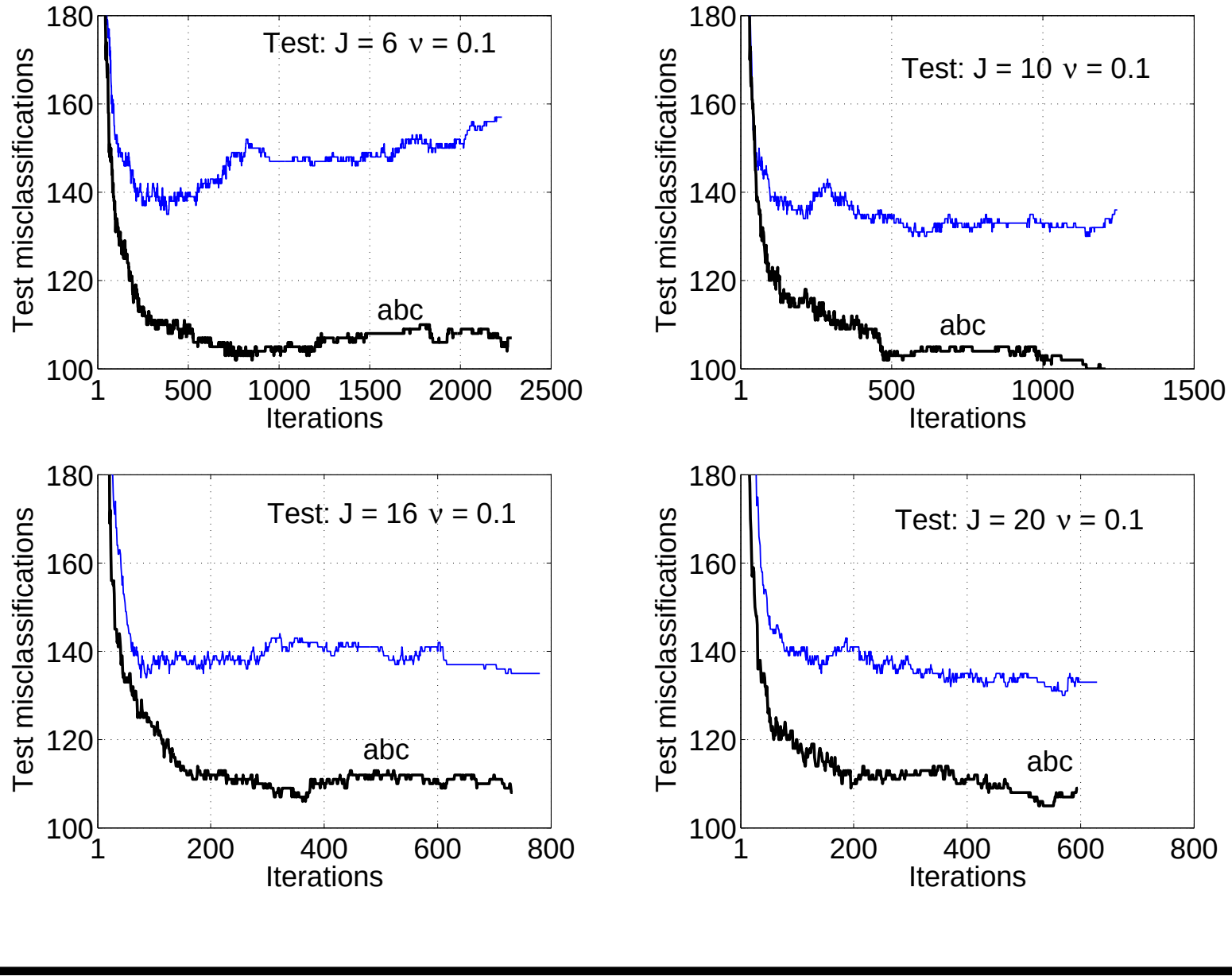

Test: J = 6 ν = 0.1
Test misclassifications
Iterations
abc
Test: J = 10 ν = 0.1
Test misclassifications
Iterations
abc
Test: J = 16 ν = 0.1
Test misclassifications
Iterations
abc
Test: J = 20 ν = 0.1
Test misclassifications
Iterations
abc

Experiments: Pendigits Data Set

	mart			
	$\nu = 0.04$	$\nu = 0.06$	$\nu = 0.08$	$\nu = 0.1$
$J = 4$	144 [145]	145 [146]	145 [144]	143 [142]
$J = 6$	135 [139]	135 [140]	143 [137]	135 [138]
$J = 8$	133 [133]	130 [132]	129 [133]	128 [134]
$J = 10$	132 [132]	129 [128]	127 [128]	130 [132]
$J = 12$	136 [134]	134 [134]	135 [140]	133 [134]
$J = 14$	129 [126]	131 [131]	130 [133]	133 [131]
$J = 16$	129 [127]	130 [129]	133 [132]	134 [126]
$J = 18$	132 [129]	130 [129]	126 [128]	133 [131]
$J = 20$	130 [129]	125 [125]	126 [130]	130 [128]

Experiments: Pendigits Data Set

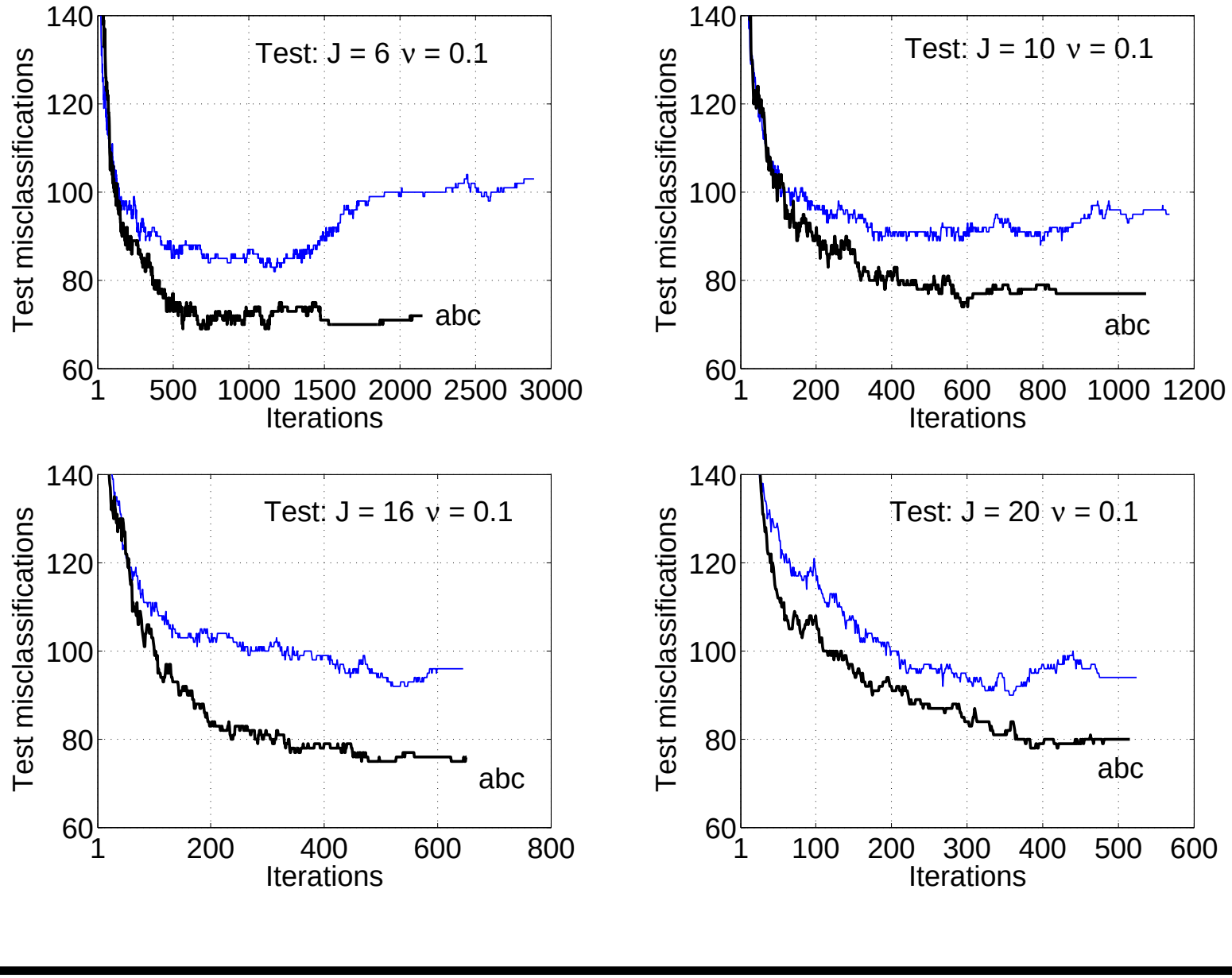
	abc-mart			
	$\nu = 0.04$	$\nu = 0.06$	$\nu = 0.08$	$\nu = 0.1$
$J = 4$	109 (24.3)	106 (26.9)	106 (26.9)	107 (25.2)
$J = 6$	109 (19.3)	105 (22.2)	104 (27.3)	102 (24.4)
$J = 8$	105 (20.5)	101 (22.3)	104 (19.4)	104 (18.8)
$J = 10$	102 (17.7)	102 (20.9)	102 (19.7)	100 (23.1)
$J = 12$	101 (25.7)	103 (23.1)	103 (23.7)	105 (21.1)
$J = 14$	105 (18.6)	102 (22.1)	102 (21.5)	102 (23.3)
$J = 16$	109 (15.5)	107 (17.7)	106 (20.3)	106 (20.9)
$J = 18$	110 (16.7)	106 (18.5)	105 (16.7)	105 (21.1)
$J = 20$	109 (16.2)	105 (16.0)	107 (15.1)	105 (19.2)


Test: J = 6 ν = 0.1
Test misclassifications
Iterations
abc
Test: J = 10 ν = 0.1
Test misclassifications
Iterations
abc
Test: J = 16 ν = 0.1
Test misclassifications
Iterations
abc
Test: J = 20 ν = 0.1
Test misclassifications
Iterations
abc

Experiments: Isolet Data Set

high-dimensional, 617 features

	mart	abc-mart
	$\nu = 0.1$	$\nu = 0.1$
$J = 4$	80 [86]	64 (20.0)
$J = 6$	84 [86]	67 (20.2)
$J = 8$	84 [88]	72 (14.3)
$J = 10$	82 [83]	74 (9.8)
$J = 12$	91 [90]	74 (18.7)
$J = 14$	95 [94]	74 (22.1)
$J = 16$	94 [92]	78 (17.0)
$J = 18$	86 [91]	78 (9.3)
$J = 20$	87 [94]	78 (10.3)

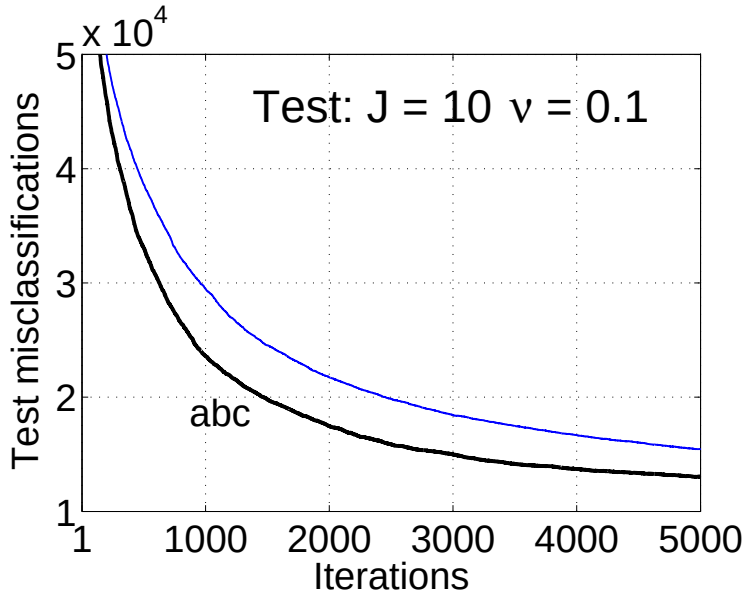

Test: J = 6 ν = 0.1
Test misclassifications
Iterations
abc
Test: J = 10 ν = 0.1
Test misclassifications
Iterations
abc
Test: J = 16 ν = 0.1
Test misclassifications
Iterations
abc
Test: J = 20 ν = 0.1
Test misclassifications
Iterations
abc

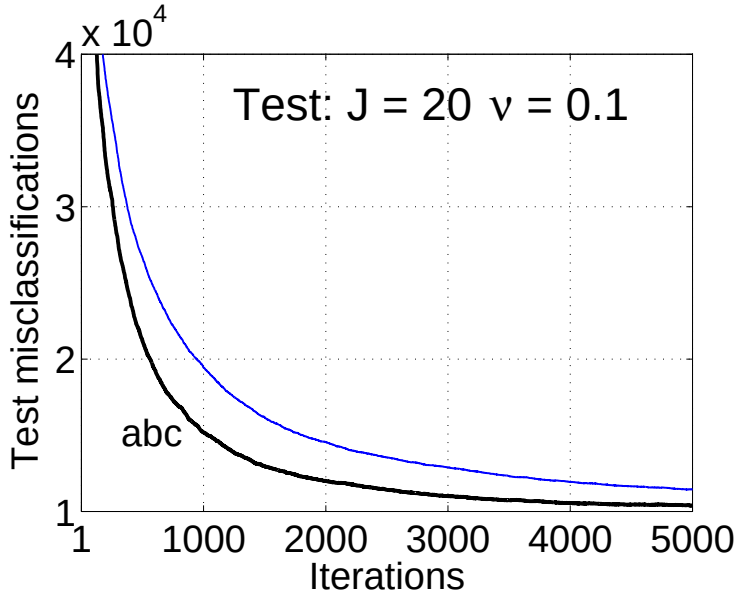
Experiments: Covertype Data Set

ν	M	J	mart	abc-mart
0.1	1000	10	29456 [29196]	23577 (20.0)
0.1	1000	20	19190 [19438]	15362 (19.6)
0.1	2000	10	21774 [21698]	17479 (19.7)
0.1	2000	20	14505 [14665]	12045 (17.0)
0.1	3000	10	18494 [18444]	14984 (19.0)
0.1	3000	20	12740 [12893]	11087 (13.0)
0.1	5000	10	15450 [15429]	13018 (15.7)
0.1	5000	20	11350 [11524]	10420 (8.2)

[11524] = result from Friedman's MART program.

(20.0) = relative improvement R_{err} (%).


x 10^4
Test misclassifications
Test: J = 10 ν = 0.1
abc
Iterations
1
2
3
4
5
1000
2000
3000
4000
5000


x 10^4
Test misclassifications
Test: J = 20 ν = 0.1
abc
Iterations
1
2
3
4
1000
2000
3000
4000
5000

Conclusion

ABC-Boost **Adaptive Base Class Boost**

- Formulate the multi-class boosting algorithm by assuming a **base** class.
- **Adaptively** select the base class at each boosting iteration.
- Considerable empirical improvements.